Part 2 - Data Structure Selection [20 Points]

Each of the following two questions describe a particular use-case or application with a need that could be satisfied by the data structures we have discussed in class so far. **For each question you must state 3 things:** the ADT from class that best fits the usage pattern described, a specific data structure implementing that ADT that best meets the performance needs, and in at most one sentence give a reason why that data structure is the best choice.

1. Alice is working on processing a bunch of her old photos by writing a program that automatically applies filters and touch-ups to batches of photos all at once. Through some experimentation, she has found that working in batches of size 100 has the best performance. What ADT can be used to store these batches of photos, and what data structure should she use to implement that ADT and why? [10 points]

   [3 points] ADT: Sequence or Seq (1 point for queue or buffer)
   [3 points] Data Structure: Array (1 point for ArrayBuffer)
   [4 points] Reason: We have fixed size of 100, so we don't need the ability to add/remove, and there are advantages to having memory be contiguous. Updates are O(1). No extra memory overhead for Array

2. Bob is a user of popular social media site TwitTok, and he wants to learn more about how many people his posts have reasonable potential to reach. To do this, he downloaded data about all of the sites users and who they follow (the site's privacy is awful). He then plans to search through this data to find out how many people on the site are at most 3 steps away from him (people who follow him are one step away, people who follow someone one step away from him are two steps away, etc). What ADT would help him organize the data in a searchable way, what data structure should he use to implement that ADT, and what makes that data structure the best choice? [10 points]

   [3 points] ADT: Graph (1 point if mentioning queue for BFS)
   [3 points] Data Structure: Adjacency List
   [4 points] Reason: The runtime to search through the graph is smaller than for edge lists and adjacency matrices since each vertex stores its incident edges

Part 3 - Stacks and Queues [17 Points]

For questions in this part, consider the following code:

```
val seq = new MysterySequence()
seq.addSomething("S")
seq.addSomething("P")
seq.addSomething("A")
seq.addSomething("C")
print(seq.removeSomething())
print(seq.removeSomething())
print(seq.removeSomething())
seq.addSomething("E")
print(seq.removeSomething())
seq.addSomething("N")
print(seq.removeSomething())
```

3. What is printed if `MysterySequence` is a queue, and `addSomething` and `removeSomething` are `enqueue` and `dequeue` respectively? What are the contents of the queue after this code is run? [5 points]

   [4 points] SPACE, [1 point] remaining contents "N"

4. What is printed if `MysterySequence` is a stack, and `addSomething` and `removeSomething` are `push` and `pop` respectively? What are the contents of the stack after this code is run? [5 points]

   [4 points] CAPEN, [1 point] remaining contents "S"

5. State the unqualified and amortized runtimes of `addSomething` and `removeSomething` for `ArrayBuffer` and `LinkedList` implementations of `MysterySequence`. [7 points]

	ArrayBuffer		**LinkedList**	
	Unqualified	**Amortized**	**Unqualified**	**Amortized**
`addSomething`	**Θ(n)**	**O(1)**	**Θ(1)**	**Θ(1)**
`removeSomething`	**Θ(1)**	**Θ(1)**	**Θ(1)**	**Θ(1)**

[1 point] per correct answer (max of 7)

Part 4 - Arrays and Linked Lists [20 Points]

For questions in this part, consider the following code, in which `array` is an `ArrayBuffer[String]`, and `list` is a `LinkedList[String]`:

```
array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")
```

6. Assume we don't know anything about the value of `x`. What is the unqualified worst-case runtime for the call inserting `"foo"` in the above code if we assume that the underlying array in our `ArrayBuffer` is not full? How does your answer change if we cannot assume that the underlying array is not full? [5 points]

   [2 points] *O*(n), [3 points] without the assumption it is still *O*(n)

7. What is the unqualified worst-case runtime for the call inserting `"bar"` in the above code if we assume that the underlying array in our `ArrayBuffer` is not full? How does your answer change if we cannot assume that the underlying array is not full? [5 points]

   [2 points] *O*(1), [3 points] without the assumption it becomes *O*(n)

8. What is the unqualified worst-case runtime for the call inserting `"baz"` in the above code if list is a singly linked list? [5 points]

   [5 points] *O*(n)

9. What is the unqualified worst-case runtime for the call inserting `"baz"` in the above code if list is a doubly linked list? [5 points]

   [5 points] *O*(1)

Part 5 - Short Answer [10 Points]

10. In one or two sentences, describe the difference between an ADT and a data structure. Then name 2 ADTs and 2 data structures we have described in class. [5 points]

    [1 point] ADTs just describe what you can do with the data, the data structure is the actual implementation of those capabilities.
    [2 points] ADTS: Seq, Buffer, Stack, Queue, Graph
    [2 points] Data Structures: Array, ArrayBuffer, LinkedList, EdgeList, AdjList, AdjMatrix

11. Consider the following runtime function for a recursive algorithm: [5 points]

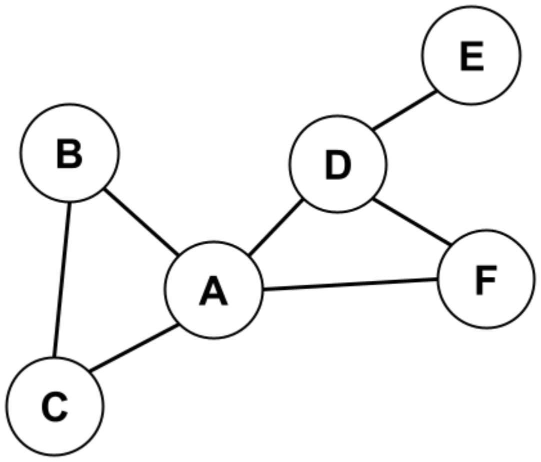
$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$

    If we hypothesize that the runtime of this recursive algorithm is ***O*(*n* log(*n*))** state inequality we must prove for the base case, and state the inequality we must use as the assumption for the inductive case.

    [2 points] Base: T(1) ≤ c * 1 * log(1)

    [3 points] Assumption: T(n/2) ≤ c * n/2 * log(n/2)

Part 6 - Graphs [15 Points]



12. Draw below a spanning subtree of the above graph that could be found using a depth-first search starting at vertex **A**: [15 points]

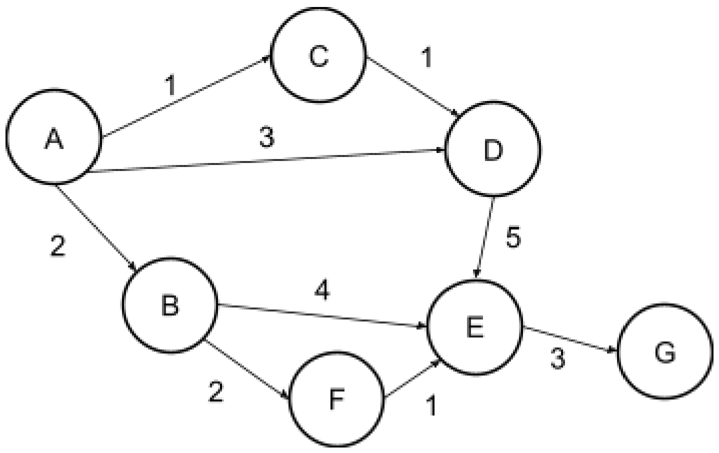
[5 points] The drawn subtree must be a spanning subgraph (contain all nodes),
[5 points] a tree (no cycles),
[5 points] and for DFS ordering it must include edge BC and edge DF

Part 7 - Extra Credit [5 Points]

13. Is it possible to have a function, $f(n)$, that is in both ***O*(*n*^{2})** and **Ω(log *n*)**?
    If so, give an example.

    [1 point] Yes. [4 points] Many possible examples: n^2, log(n), n, case functions, etc.

Part 2 - Graphs (PA3) [20 Points]



Imagine we want to find the shortest paths in the above weighted graph starting from A.

1. In PA3, what **ADT** was used to organize the nodes in the work list to ensure we traversed them in the correct order to compute the shortest path? [5 points]
   RUBRIC: 5 points if correct (priority queue, or heap). 2 points if queue.

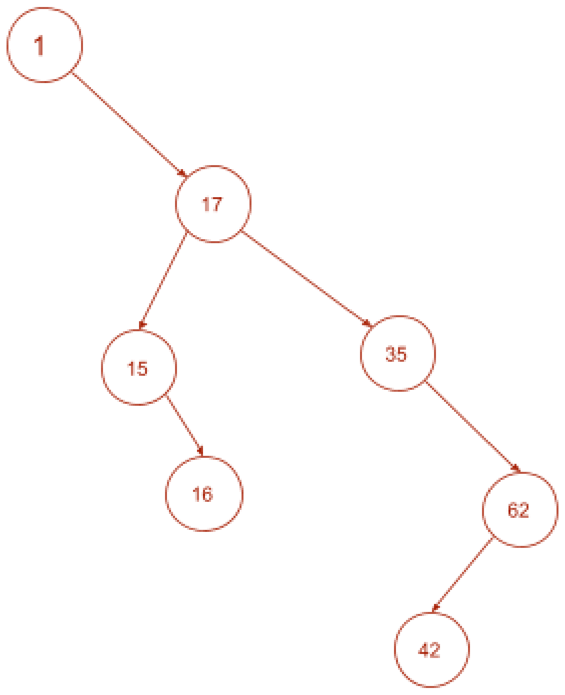
2. At each step of the algorithm we dequeue a `(vertex, distance)` pair from our work list. For example, the first dequeue for the above graph would be `(A, 0)`. Write out the sequence of dequeue operations starting with `(A,0)`, and ending when the node **F** is dequeued. **Break ties in alphabetical order.** [10 points]
   RUBRIC: 2 points per correct dequeue after (A,0) [only 1 point per dequeue that is a correct element but in the wrong order]
   (A,0), (C,1), (B,2), (D,2), (D,3), (F,4)

3. Write out the **`(vertex, distance)`** pairs that are still in the work list **after F is processed**, or write none if no pairs will remain in the work list. [5 points]
   RUBRIC: 2 points each for mentioning (E,6), (E,7). 1 for (E,5)
   (E,5), (E,6), (E,7)

Part 4 - Trees [20 Points]

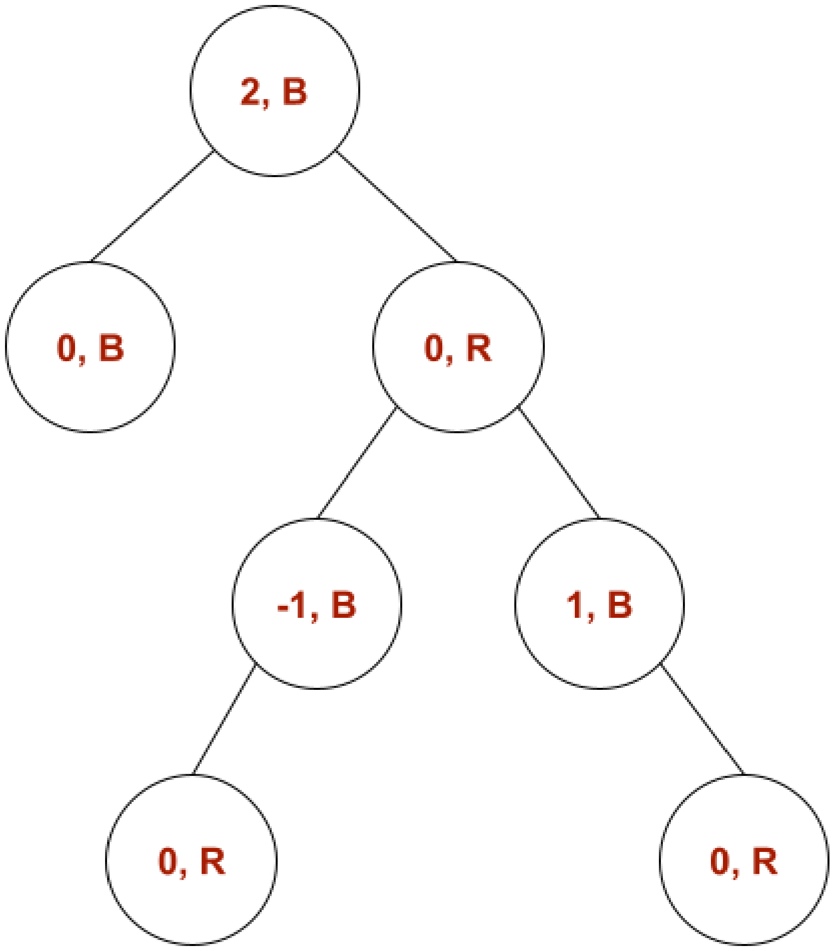
4. Draw the BST (regular BST, not AVL or Red-Black) that results from inserting the following elements in the order listed: [7 points]
   RUBRIC: 1 point per correctly inserted node

   1, 17, 35, 15, 16, 62, 42



5. Give one example of a rotation that would **decrease** the height of the tree in the previous question. [3 points]
   RUBRIC: 3 points for correct rotation (rotate left around 1 or 35), or if they give a correct rotation for the tree they drew.

6. The following tree is a Red-Black tree. For each node, label it with its balance factor, as well as a color (red or black) that would lead to a valid coloring according to the Red-Black tree constraints. [7 points]
   RUBRIC: 0.5 points per correct label (if only thing wrong is sign, deduct 1 total)



7. Is the above tree a valid AVL tree? Why or why not? [3 points]
   RUBRIC: 1 point for no. 2 for pointing out valid reason why.
   No. The balance factor of the root is 2.